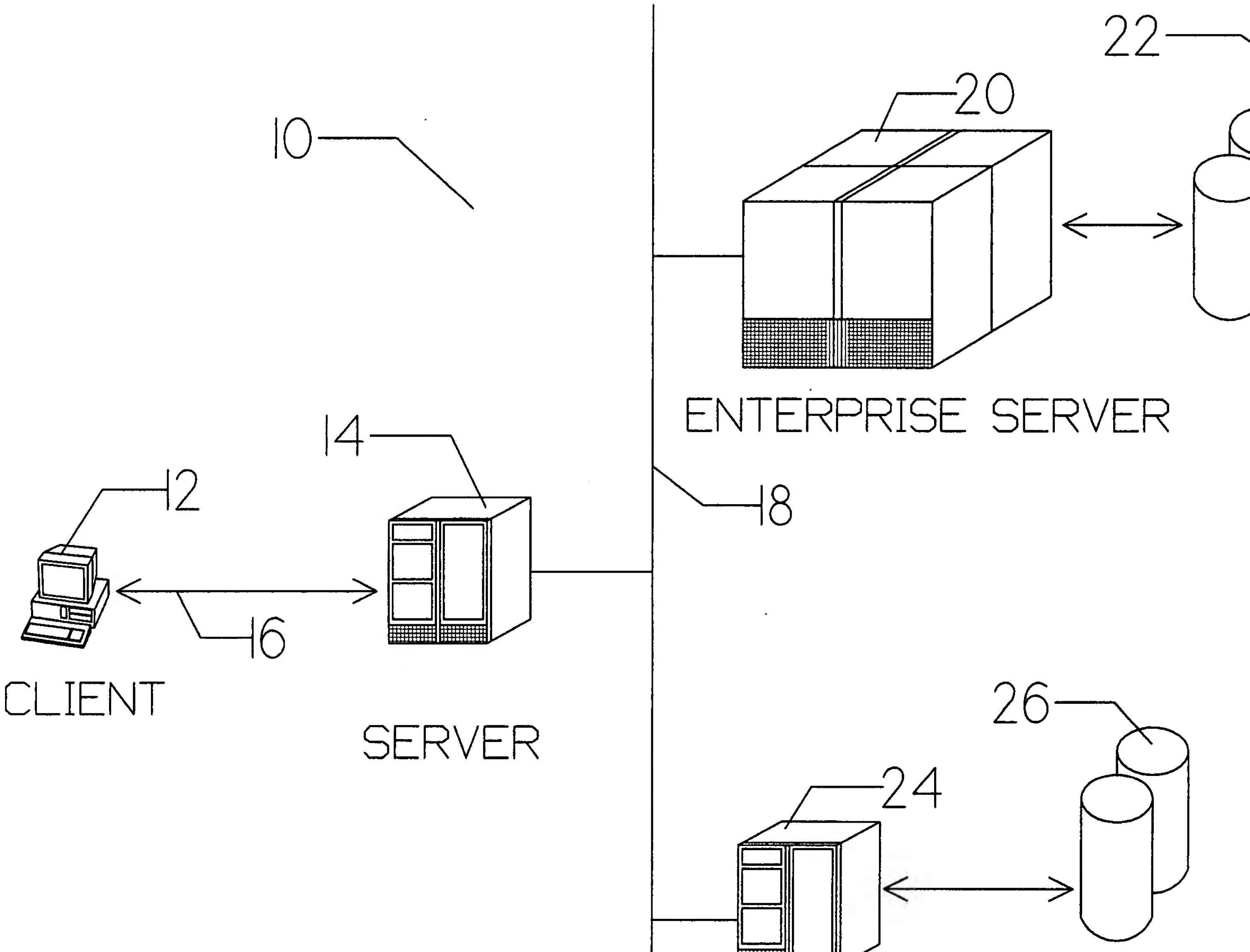

10
22
20
ENTERPRISE SERVER
14
12
18
16
CLIENT
SERVER
26
24
DEPARTMENTAL SERVER

FIG. 1

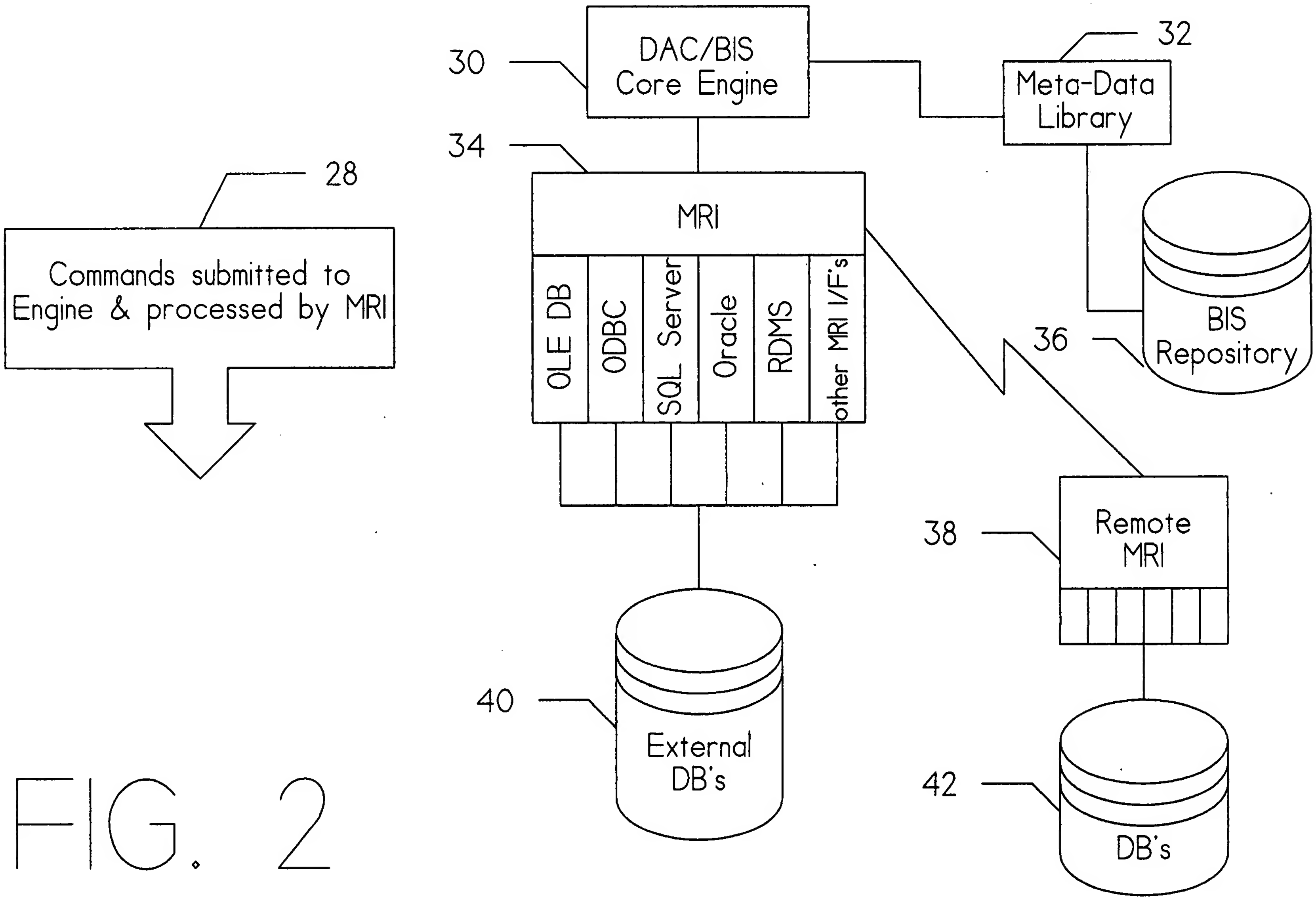
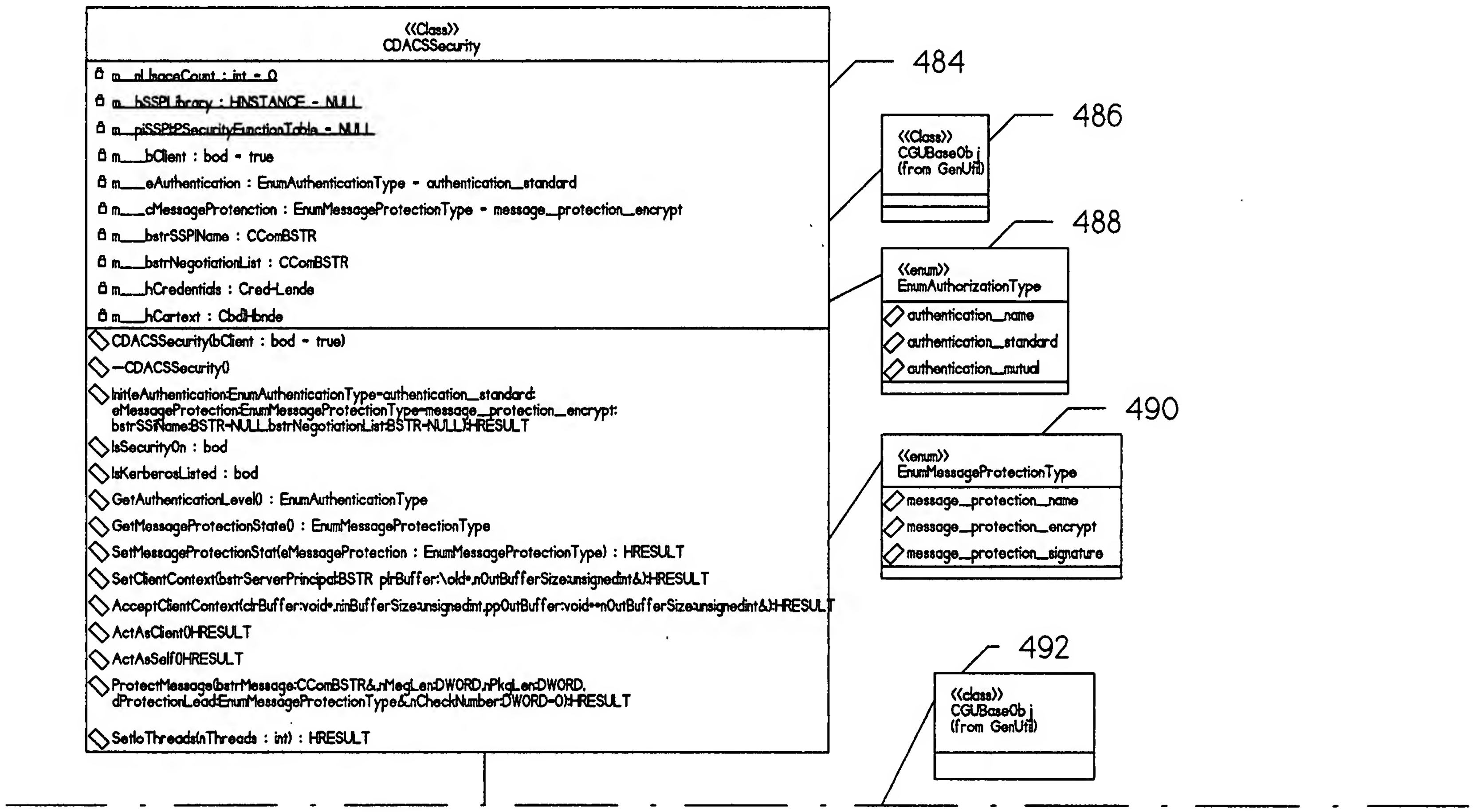

28
Commands submitted to Engine & processed by MRI
30
DAC/BIS Core Engine
32
Meta-Data Library
34
MRI
OLE DB
ODBC
SQL Server
Oracle
RDMS
other MRI I/F's
36
BIS Repository
38
Remote MRI
40
External DB's
42
DB's

FIG. 2



<<Class>>
CDACSSecurity
m_nUsaceCount : int = 0
m_hSSPILibrary : HINSTANCE = NULL
m_piSSPIPSecurityFunctionTable = NULL
m___bClient : bod = true
m___eAuthentication : EnumAuthenticationType = authentication_standard
m___cMessageProtenction : EnumMessageProtectionType = message_protection_encrypt
m___bstrSSPIName : CComBSTR
m___bstrNegotiationList : CComBSTR
m___hCredentials : Cred-Lende
m___hCartext : CbdHbnde
CDACSSecurity(bClient : bod = true)
~CDACSSecurity()
Init(eAuthentication:EnumAuthenticationType=authentication_standard:
eMessageProtection:EnumMessageProtectionType=message_protection_encrypt:
bstrSSPName:BSTR=NULL,bstrNegotiationList:BSTR=NULL):HRESULT
IsSecurityOn : bod
IsKerberosListed : bod
GetAuthenticationLevel() : EnumAuthenticationType
GetMessageProtectionState() : EnumMessageProtectionType
SetMessageProtectionStat(eMessageProtection : EnumMessageProtectionType) : HRESULT
SetClientContext(bstrServerPrincipal:BSTR plrBuffer:\old*,nOutBufferSize:unsignedint&):HRESULT
AcceptClientContext(clrBuffer:void*,ninBufferSize:unsignedint,ppOutBuffer:void**nOutBufferSize:unsignedint&):HRESULT
ActAsClient()HRESULT
ActAsSelf()HRESULT
ProtectMessage(bstrMessage:CComBSTR&,nMegLen:DWORD,nPkgLen:DWORD,
dProtectionLead:EnumMessageProtectionType&,nCheckNumber:DWORD=0):HRESULT
SetIoThreads(nThreads : int) : HRESULT
484
<<Class>>
CGUBaseObj
(from GenUtil)
486
<<enum>>
EnumAuthorizationType
authentication_name
authentication_standard
authentication_mutual
488
<<enum>>
EnumMessageProtectionType
message_protection_name
message_protection_encrypt
message_protection_signature
490
<<class>>
CGUBaseObj
(from GenUtil)
492

FIG. 3A

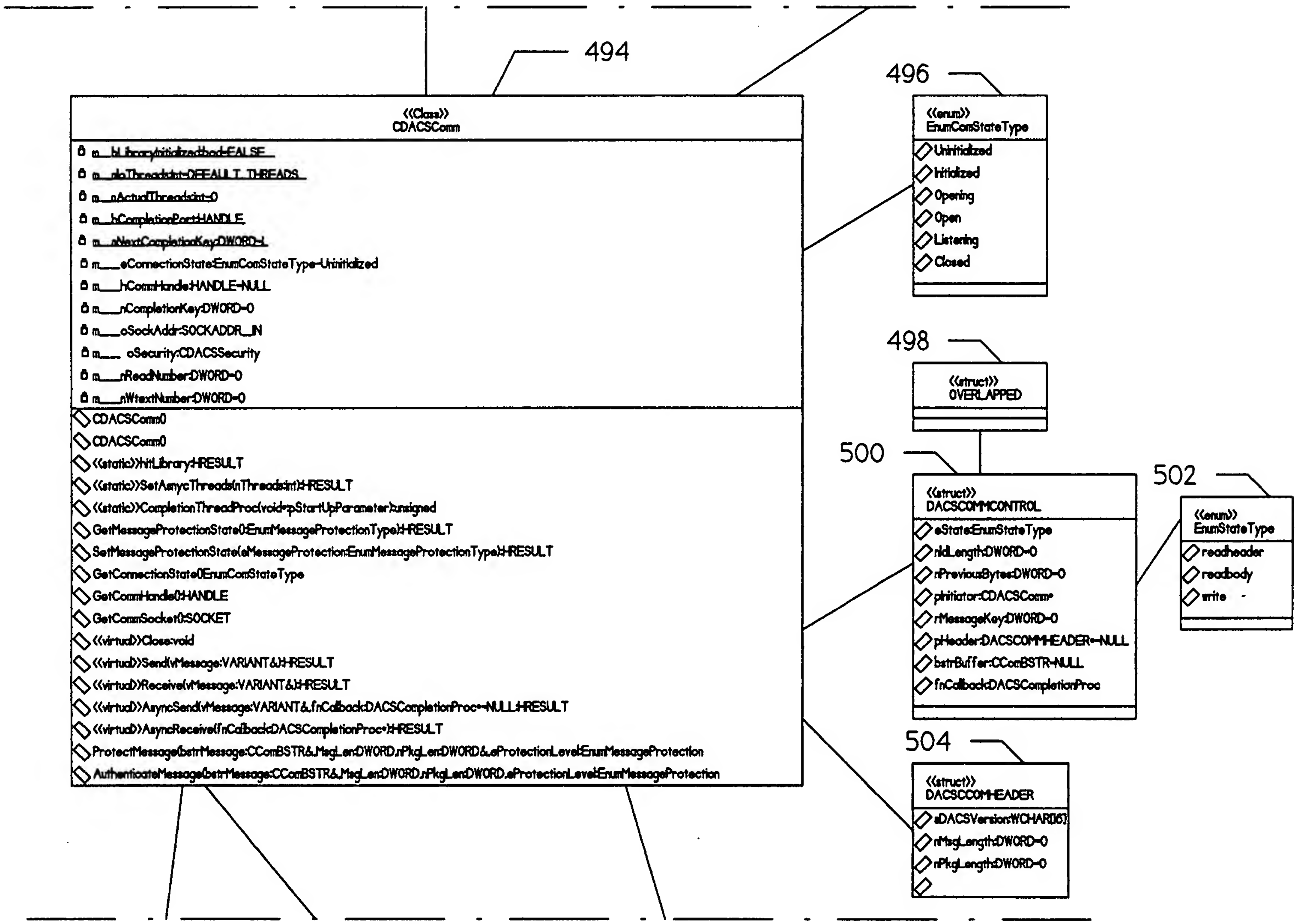

494
<<Class>>
CDACSComm
m__bLibraryInitialized:bool=FALSE
m__nloThreadsInt=DEFAULT_THREADS
m__nActualThreads:int=0
m__hCompletionPort:HANDLE
m__nNextCompletionKey:DWORD=1
m___eConnectionState:EnumComStateType=Uninitialized
m___hCommHandle:HANDLE=NULL
m___nCompletionKey:DWORD=0
m___oSockAddr:SOCKADDR_IN
m___ oSecurity:CDACSSecurity
m___nReadNumber:DWORD=0
m___nWtextNumber:DWORD=0
CDACSComm0
CDACSComm0
<<static>>InitLibrary:HRESULT
<<static>>SetAsyncThreads(nThreads:int):HRESULT
<<static>>CompletionThreadProc(void*pStartUpParameter):unsigned
GetMessageProtectionState0:EnumMessageProtectionType):HRESULT
SetMessageProtectionState(eMessageProtection:EnumMessageProtectionType):HRESULT
GetConnectionState0EnumComStateType
GetCommHandle0:HANDLE
GetCommSocket0:SOCKET
<<virtual>>Close:void
<<virtual>>Send(vMessage:VARIANT&):HRESULT
<<virtual>>Receive(vMessage:VARIANT&):HRESULT
<<virtual>>AsyncSend(vMessage:VARIANT&,fnCallback:DACSCompletionProc*=NULL):HRESULT
<<virtual>>AsyncReceive(fnCallback:DACSCompletionProc*):HRESULT
ProtectMessage(bstrMessage:CComBSTR&,MsgLen:DWORD,nPkgLen:DWORD&,eProtectionLevel:EnumMessageProtection
AuthenticateMessage(bstrMessage:CComBSTR&,MsgLen:DWORD,nPkgLen:DWORD,eProtectionLevel:EnumMessageProtection
496
<<enum>>
EnumComStateType
Uninitialized
Initialized
Opening
Open
Listening
Closed
498
<<struct>>
OVERLAPPED
500
<<struct>>
DACSCOMMCONTROL
eState:EnumStateType
nldLength:DWORD=0
nPreviousBytes:DWORD=0
pInitiator:CDACSComm*
nMessageKey:DWORD=0
pHeader:DACSCOMMHEADER*=NULL
bstrBuffer:CComBSTR=NULL
fnCallback:DACSCompletionProc
502
<<enum>>
EnumStateType
readheader
readbody
write
504
<<struct>>
DACSCCOMHEADER
sDACSVersion:WCHAR[16]
nMsgLength:DWORD=0
nPkgLength:DWORD=0

FIG. 3B

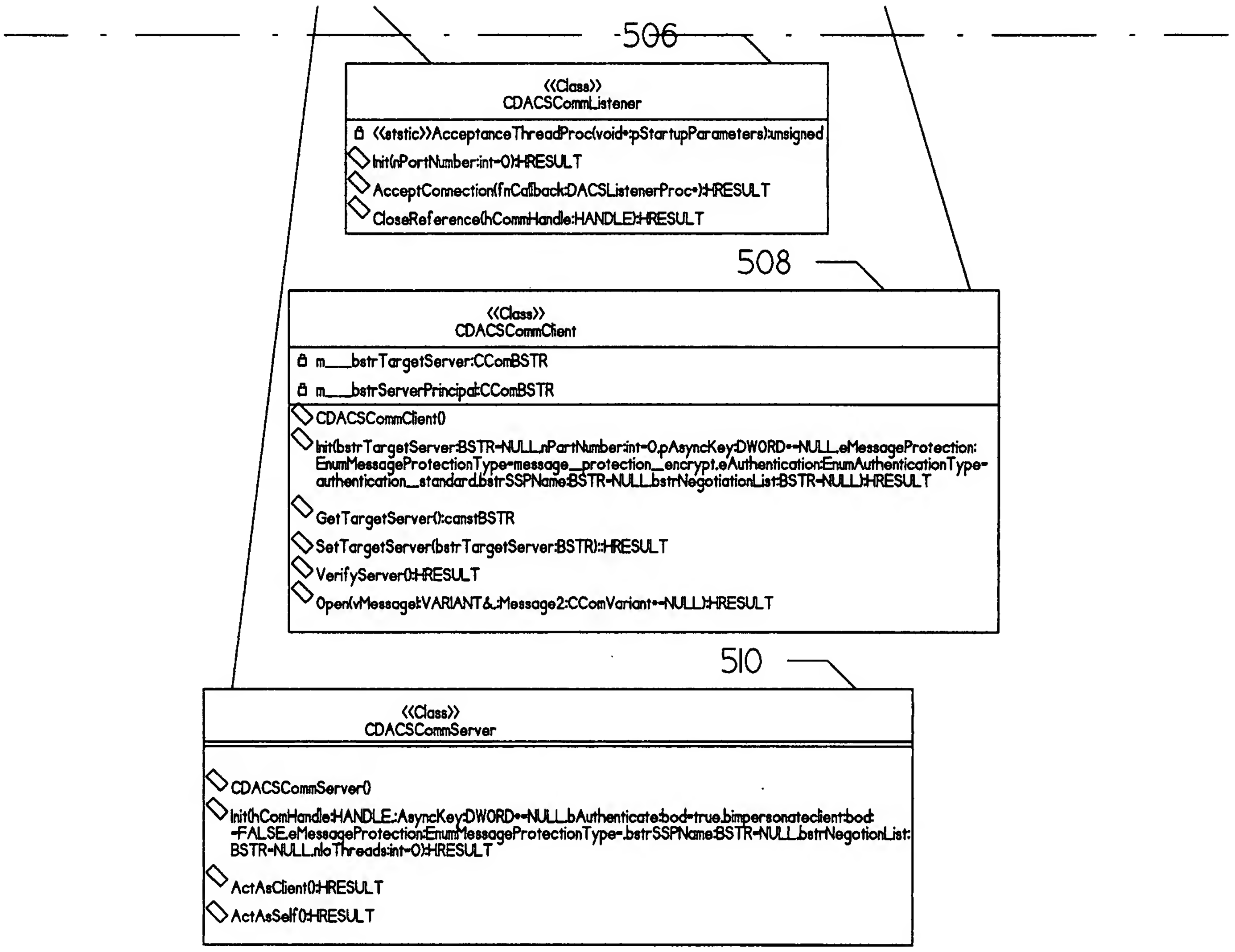

506
<<Class>>
CDACSCommListener
<<ststic>>AcceptanceThreadProc(void*pStartupParameters):unsigned
Init(nPortNumber:int-0):HRESULT
AcceptConnection(fnCallback:DACSListenerProc*):HRESULT
CloseReference(hCommHandle:HANDLE):HRESULT
508
<<Class>>
CDACSCommClient
m___bstrTargetServer:CComBSTR
m___bstrServerPrincipal:CComBSTR
CDACSCommClient()
Init(bstrTargetServer:BSTR-NULL.nPartNumber:int-0.pAsyncKey:DWORD*-NULL.eMessageProtection: EnumMessageProtectionType-message__protection__encrypt.eAuthentication:EnumAuthenticationType-authentication__standard.bstrSSPName:BSTR-NULL.bstrNegotiationList:BSTR-NULL):HRESULT
GetTargetServer():constBSTR
SetTargetServer(bstrTargetServer:BSTR)::HRESULT
VerifyServer():HRESULT
Open(vMessage1:VARIANT&.:Message2:CComVariant*-NULL):HRESULT
510
<<Class>>
CDACSCommServer
CDACSCommServer()
Init(hComHandle:HANDLE.:AsyncKey:DWORD*-NULL.bAuthenticate:bool-true.bimpersonateclient:bool -FALSE.eMessageProtection:EnumMessageProtectionType-.bstrSSPName:BSTR-NULL.bstrNegotionList: BSTR-NULL.nloThreads:int-0):HRESULT
ActAsClient():HRESULT
ActAsSelf():HRESULT

FIG. 3C

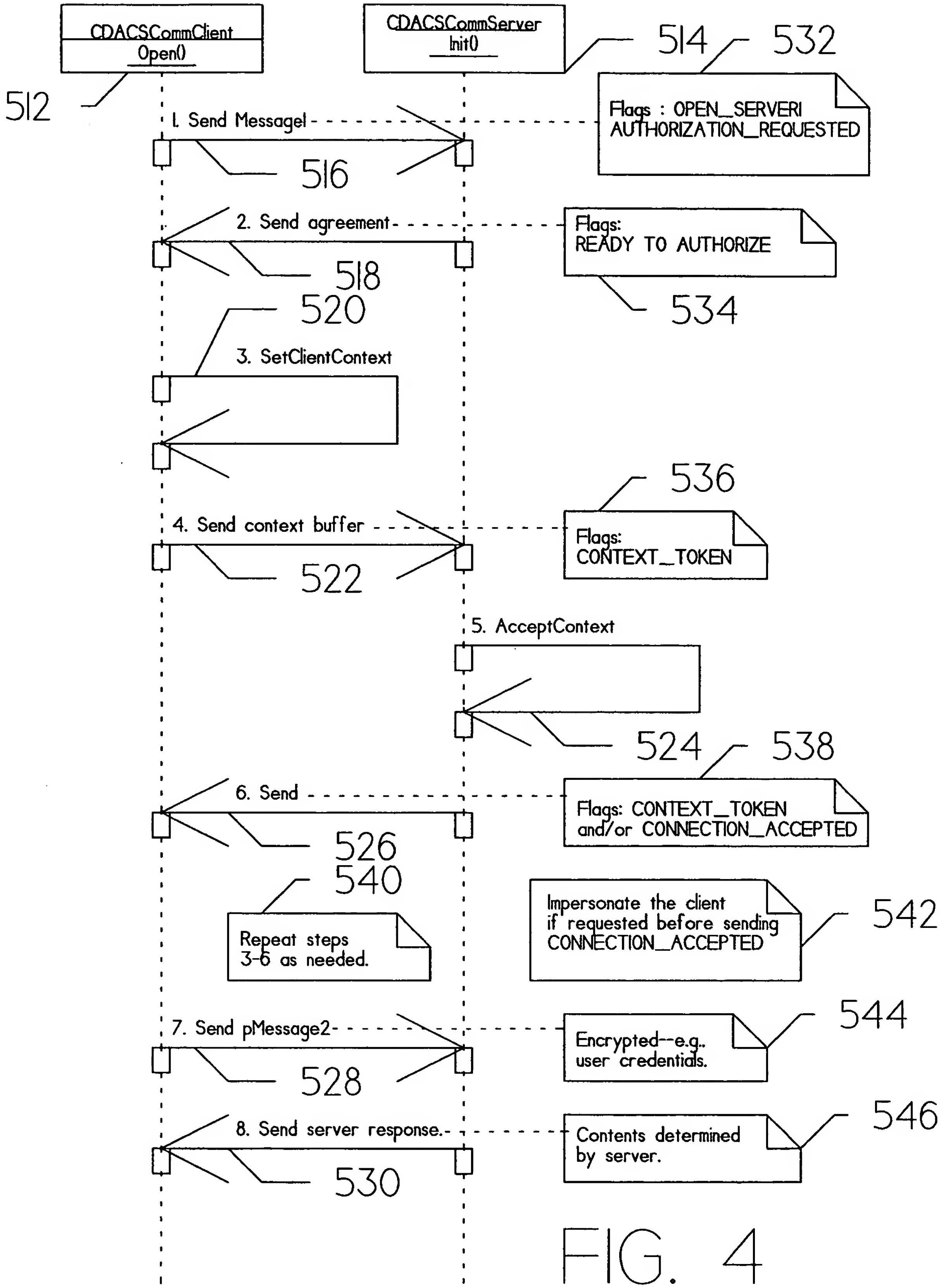

CDACSCommClient
Open()
CDACSCommServer
Init()
514
532
512
Flags : OPEN_SERVERI
AUTHORIZATION_REQUESTED
1. Send Message1
516
2. Send agreement
Flags:
READY TO AUTHORIZE
518
520
534
3. SetClientContext
536
4. Send context buffer
Flags:
CONTEXT_TOKEN
522
5. AcceptContext
524
538
6. Send
Flags: CONTEXT_TOKEN
and/or CONNECTION_ACCEPTED
526
540
Repeat steps
3-6 as needed.
Impersonate the client
if requested before sending
CONNECTION_ACCEPTED
542
544
7. Send pMessage2
Encrypted--e.g..
user credentials.
528
546
8. Send server response.
Contents determined
by server.
530

FIG. 4

Message #	Description	Fig. 16 Reference
1	After the connection has been accepted. send the caller's initial message. This message is not encrypted.	516
2	If authorization is requested and agreed by the server. tell client that the server is ready for its security context. Otherwise. go to step 6.	518
3	Call the SSPI InitializeSecurityContext function to get a context token.	520
4	Send the client context token to the server.	522
5	Call the SSPI AcceptSecurityContext function with the buffer received from the client.	524
6	THis step depends on the SSP. Set the CONTEXT_TOKEN flag if AcceptSecurityContext provided an output buffer. If it returned SEC_E_OK. impersonate the client if so requested by the server consumer. and set the CONNECTION_ACCEPTED flag.	526
7	Since vMessage1 is sent unencrypted. the client may provide a second message to send encrypted.	528
8	Open attaches the server response to the pMessage2 parameter as output. The consumer client should check both the return value from Open and the contents of pMessage2 if used.	530

FIG. 5